## Remarks

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks. Because claims 1-5, 7-12, 15-17, and 22-28 are allowed by the Office, they have not been discussed in this Response.

### Patentability Over CORBA in view of Steinman and Coskun

The Office has asserted a rejection of claims 13, and 18-21, under 35 U.S.C. § 103(a) over "The Common Object Request Broker: Architecture and Specification, CORBA," Revision 2.0, July 1995 ("CORBA") in view of Steinman, J., "Incremented State Saving in SPEEDS Using C++,"" Proceedings of the 1993 Winter Simulation Conference ("Steinman") and Coskun, U. S. Patent No. 5,764,958 ("Coskun"). Applicants respectfully traverse.

**Claim 13**

Claim 13 is generally directed to a method of enhancing scalability of server applications comprising "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete."

Specifically, claim 13 recites,

> 13. In a computer, a method of encapsulating state of processing work for a client by a server application in a component with improved scalability, comprising:
>
> encapsulating function code and a processing state for the work in a component;
>
> providing a reference through an operating service for a client to call the function code of the component to initiate processing of the work by the component;
>
> receiving an indication from the component that the work by the component is complete; and
>
> *discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete.*
>
> (Emphasis Added).

Thus, the component can indicate discarding its own state upon completion of the work without any indication from the client that the component's work is complete.

In considering whether a CORBA-Steinman-Coskun combination teaches or suggests the recited language, Applicants respectfully submit that it is helpful to ask three questions--first, what component's state is discarded, second, what component indicated completion of work, and third, did the client indicate that work is complete? In claim 13, the component's state is discarded ("discarding the processing state of the component"), the component indicated that work is complete ("responsive to the component indicating completion of the work") and the client of the component did not indicate that the component's work is complete ("before receiving any indication from the client that the component's work is complete.")

Applicants respectfully submit that a CORBA-Coskun-Steinman combination fails to teach or suggest "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete."

The Examiner admits that CORBA fails to teach or suggest the recited arrangement. *See* Office Action, mailed July 8, 2003, page 4, ¶ 2. Second, the Examiner does not allege that Steinman teaches or suggests the recited arrangement. Since the references when combined "must teach or suggest all the claim limitations," Coskun must teach or suggest the recited arrangement.

The Examiner asserts that the following Coskun passages, disclose the recited arrangement,

> It is desirable to have a mechanism that can add roles to objects dynamically, depending on the context of the object while limiting the overhead requirements. *Col. 1, ll 56-58.*
>
> ...
>
> For example, when a program is started, a person object will only have a student object. Before calling a function requiring teacher characteristics, a teacher role is added to the person object. When the function call returns, the teacher role is deleted from the person object if the role is no longer needed. *Col. 2, ll 14-19.*
>
> ...
>
> It will also be apparent to those skilled in the art that when an application program is started requiring only the person object 82, and later requires teacher characteristics, the teacher object 86, may be loaded, and when the function call returns, the teacher object 86, may be deleted if no longer needed. *Col. 4, ll 31-36.*

In summary, Coskun describes a mechanism that adds roles dynamically and deletes roles dynamically as needed. However, Coskun fails to teach or suggest the claimed language, namely--

*who is indicating work is complete*? Specifically, since the teacher role is what is discarded in the above passage, the question must be asked—is the teacher role indicating that work is complete?

For example, in one passage cited by the Examiner, Coskun states, when "the function call returns, the teacher role is deleted from the person object if the role is no longer needed." Additionally, "the teacher object 86, may be deleted if no longer needed." Thus, the teacher object is deleted when it is no longer needed. These statements support the object oriented cannon--objects are created when they are needed, and discarded when they are no longer needed. Since objects are needed to process requests from other objects, they are discarded when no longer needed by requesting objects. Thus, when the teacher object is no longer needed by a requesting object, it is discarded. Thus, the recited passage fails to teach or suggest "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete."

In Coskun, the teacher object doesn't know or care whether or not it is still needed. It is agnostic about its own lifespan. It makes no decisions or requests that could in anyway affect its own lifespan. In Coskun, requests or decisions about an objects lifespan are made outside the object.

Nowhere does Coskun state that the teacher object is destroyed response to its own indication. The claim language requires a component being discarded responsive to the component's own indication that work is complete, and before any indication from the client that work is complete. Without showing this element, no combination can teach or suggest claim 13.

For at least this reason claim 13 should be allowed. Such action is respectfully requested.

**Claim 18**

Claim 18 is generally directed to a system service comprising "destroying the processing state of the application component responsive to the indication from the application component that processing by the application component of the work is complete." Specifically, amended claim 18 recites,

> 18. In a computer, a system service for providing an execution environment for scalable application components, comprising:
>
> code responsive to a request from a client program to create an application component for returning to the client program a reference through the system service to the application component;
>
> code responsive to a call from the client program using the reference for initiating processing of work by the application

> component, the application component producing a processing state during processing the work;
>
> code for receiving an indication from the application component that processing by the application component of the work is complete; and
>
> *code for destroying the processing state of the application component responsive to the indication from the application component that processing by the application component of the work is complete and without action from the client program.*
> (Emphasis Added).

In considering whether a CORBA-Steinman-Coskun combination teaches or suggests the recited language, Applicants respectfully submit that it is helpful to ask three questions--first, what component's state is destroyed, second, responsive to what component's indication, and third, without action by the client? In claim 18, (1) the application component's state is destroyed, (2) responsive to the indication from the application component that processing by the application component of the work is complete and (3) without action from the client program.

First, the Examiner admits that CORBA fails to teach or suggest the recited arrangement. *See* Office Action, mailed July 8, 2003, page 4, ¶ 2. Second, the Examiner does not allege that Steinman teaches or suggests the recited arrangement. Since the references when combined "must teach or suggest all the claim limitations," Coskun must teach or suggest the recited arrangement.

The Examiner asserts that the following Coskun passages, disclose the recited arrangement,

> It is desirable to have a mechanism that can add roles to objects dynamically, depending on the context of the object while limiting the overhead requirements. *Col. 1, ll 56-58.*
>
> ...
>
> For example, when a program is started, a person object will only have a student object. Before calling a function requiring teacher characteristics, a teacher role is added to the person object. When the function call returns, the teacher role is deleted from the person object if the role is no longer needed. *Col. 2, ll 14-19.*
>
> ...
>
> It will also be apparent to those skilled in the art that when an application program is started requiring only the person object 82, and later requires teacher characteristics, the teacher object 86, may be loaded, and when the function call returns, the teacher object 86, may be deleted if no longer needed. *Col. 4, ll 31-36.*

In summary, Coskun describes a mechanism that adds roles dynamically and deletes roles dynamically as needed. However, Coskun fails to teach or suggest the claimed language, namely--

*who is indicating work is complete*? Specifically, since the teacher role is what is discarded in the above passage, the question must be asked—is the teacher role indicating that work is complete?

For example, in one passage cited by the Examiner, Coskun states, when "the function call returns, the teacher role is deleted from the person object if the role is no longer needed." Additionally, "the teacher object 86, may be deleted if no longer needed." Thus, the teacher object is deleted when it is no longer needed. These statements support the object oriented cannon--objects are created when they are needed, and discarded when they are no longer needed. Since objects are needed to process requests from other objects, they are discarded when no longer needed by requesting objects. Thus, when the teacher object is no longer needed by a requesting object, it is discarded. Thus, the recited passage fails to teach or suggest "destroying the processing state of the application component responsive to the indication from the application component that processing by the application component of the work is complete."

In Coskun, the teacher object doesn't know or care whether or not it is still needed. It is agnostic about its own lifespan. It makes no decisions or requests that could in anyway affect its own lifespan. In Coskun, requests or decisions about an objects lifespan are made outside the object.

Nowhere does Coskun state that the teacher object is destroyed response to its own indication. The claim language requires a component being discarded responsive to the component's own indication that work is complete, and before any indication from the client that work is complete. Without showing this element, no combination can teach or suggest claim 18.

For at least this reason claim 18 should be allowed. Such action is respectfully requested.

**Claim 19**

Claim 19 depends from claim 18. Since claim 19 depends from claim 18, it should be allowed for at least the reasons stated for claim 18. In view of the foregoing discussion claim 18, the merits of the separate patentability of dependent claim 19 not belabored at this time. Claims 19 should be allowed. Such action is respectfully requested.

**Claim 20**

Claim 20 depends from claim 18. Since claim 20 depend from claim 18, it should be allowed for at least the reasons stated for claim 18. In view of the foregoing discussion claim 18, the merits

of the separate patentability of dependent claim 20 is not belabored at this time. Claim 20 should be allowed. Such action is respectfully requested.

**Claim 21**

Claim 21 is generally directed to a method of enhancing scalability of server applications comprising an "application component having state ... destroying the state by the operating service in response to an indication from the application component without action by the client."

Specifically, claim 21 recites,

> 21. In a computer having a main memory, a method of enhancing scalability of server applications, comprising:
>
> executing an application component under control of an operating service, the application component having a state and function code for performing work responsive to method invocations from a client;
>
> maintaining the state in the main memory between the method invocations of the function code by the client in the absence of an indication from the application component that the work is complete; and
>
> *destroying the state by the operating service in response to an indication from the application component without action by the client, such that the destroyed state is not persistent. (Emphasis Added).*

In considering whether a CORBA-Steinman-Coskun combination teaches or suggests the recited language, Applicants respectfully submit that it is helpful to ask three questions--what component's state is destroyed, responsive to what component's indication, and did the client take action? In claim 21, (1) the application component's state is destroyed, (2) response to an indication from the application component, and (3) without action by the client." Applicants respectfully submit that a CORBA-Coskun-Steinman combination fails to teach or suggest "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete."

First, the Examiner admits that CORBA fails to teach or suggest the recited arrangement. *See* Office Action, mailed July 8, 2003, page 4, ¶ 2. Second, the Examiner does not allege that Steinman teaches or suggests the recited arrangement. Since the references when combined "must teach or suggest all the claim limitations," Coskun must teach or suggest the recited arrangement.

The Examiner asserts that the following Coskun passages, disclose the recited arrangement,

> It is desirable to have a mechanism that can add roles to objects dynamically, depending on the context of the object while limiting the overhead requirements. *Col. 1, ll 56-58.*
>
> ...
>
> For example, when a program is started, a person object will only have a student object. Before calling a function requiring teacher characteristics, a teacher role is added to the person object. When the function call returns, the teacher role is deleted from the person object if the role is no longer needed. *Col. 2, ll 14-19.*
>
> ...
>
> It will also be apparent to those skilled in the art that when an application program is started requiring only the person object 82, and later requires teacher characteristics, the teacher object 86, may be loaded, and when the function call returns, the teacher object 86, may be deleted if no longer needed. *Col. 4, ll 31-36.*

In summary, Coskun describes a mechanism that adds roles dynamically and deletes roles dynamically as needed. However, Coskun fails to teach or suggest the claimed language, namely—responsive to whose indication? Specifically, since the teacher role is what is destroyed in the above passage, the question must be asked—is the destruction responsive to the teacher's indication?

For example, in one passage cited by the Examiner, Coskun states, when "the function call returns, the teacher role is deleted from the person object if the role is no longer needed." Additionally, "the teacher object 86, may be deleted if no longer needed." Thus, the teacher object is deleted when it is no longer needed. These statements support the object oriented cannon--objects are created when they are needed, and discarded when they are no longer needed. Since objects are needed to process requests from other objects, they are discarded when no longer needed by requesting objects. Thus, when the teacher object is no longer needed by a requesting object, it is discarded. Thus, the recited passage fails to teach or suggest an "application component having state ... destroying the state by the operating service in response to an indication from the application component without action by the client."

In Coskun, the teacher object doesn't know or care whether or not it is still needed. It is agnostic about its own lifespan. It makes no decisions or requests that could in anyway affect its own lifespan. In Coskun, requests or decisions about an objects lifespan are made outside the object.

Nowhere does Coskun state that the teacher object is destroyed response to its own indication. The claim language requires a component being discarded responsive to the component's

own indication that work is complete, and before any indication from the client that work is complete. Without showing this element, no combination can teach or suggest claim 21.

For at least this reason claim 21 should be allowed. Such action is respectfully requested.

**Double Patenting**

The Office has asserted a rejection of claims 13, and 18-21, under the judicially created doctrine of non-statutory double patenting over claims 1-21 of Helland, U.S. Patent No. 5,890,161 ("Helland") in view of Steinman, J., "Incremented State Saving in SPEEDS Using C++,"" Proceedings of the 1993 Winter Simulation Conference ("Steinman") and Hutchinson, U. S. Patent No. 5,764,958 ("Hutchinson"). Applicants respectfully traverse.

**Claim 13**

Claim 13 is generally directed to a method of enhancing scalability of server applications comprising "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete."

Specifically, claim 13 recites,

> 13. In a computer, a method of encapsulating state of processing work for a client by a server application in a component with improved scalability, comprising:
>
> encapsulating function code and a processing state for the work in a component;
>
> providing a reference through an operating service for a client to call the function code of the component to initiate processing of the work by the component;
>
> receiving an indication from the component that the work by the component is complete; and
>
> *discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete.*
>
> (Emphasis Added).

Thus, the component can indicate discarding its own state upon completion of the work without any indication from the client that the component's work is complete.

The Examiner asserts a double patenting in light of a Helland-Steinman-Hutchinson combination. Applicants disagree. First, the recited language is not in claims 7, 13, 14, and 15 of

Helland as indicated by the Office. Rather in Helland, claim 7 recites, "receiving an indication from the application component that its work in the new transaction is complete; and upon the application component next returning from a client call, causing the new transaction to complete, claim 13 recites, "a transaction completion member function for the application component to indicate completion of its transactional work; and the run-time service operating to cause the transaction to complete upon the application component returning from a client call after calling the transaction completion member function," claim 14 recites, "the run-time service operates to cause the transaction to abort upon the application component returning from the client call after calling the set abort function," and claim 15 recites "a transaction context object for representing a context having a transaction initiated and controlled by the client."

Applicants respectfully submit that claim 13 is not recited in claims 7, 13, 14, or 15 of Helland, and further, that neither Hutchinson or Steinman teach or suggest "(1) discarding the processing state of the component (2) responsive to the component indicating completion of the work (3) before receiving any indication from the client that the component's work is complete."

In considering whether Hutchinson or Steinman teach or suggest the recited language, Applicants respectfully submit that three things must be considered--what component's state is discarded, what component indicated completion of work, and did the client indicate that work is complete? In claim 13, the component's state is discarded ("discarding the processing state of the component"), the component indicated that work is complete ("responsive to the component indicating completion of the work") and the client of the component did not indicate that the component's work is complete ("before receiving any indication from the client that the component's work is complete.")

The Office asserts that "Hutchinson teaches destroying an object's state when its transactional work is complete," however, Hutchinson fails to teach or suggest (1) what component's state is discarded, (2) what component indicated completion of work, or (3) did the client indicate that work is complete? The Office asserts the following Hutchinson passages,

> A problem resides in the fact that different programs and services often have different mechanisms for managing the mapping from a given user to a given context. For example, some database programs store all their context information for a given transaction based on the thread ID from which the database program was called. Therefore, if multiple requests for the same transaction come into the server, those requests must run on the same thread, otherwise the database program will not use the correct context. For another example, some database programs provide

means to allow you to take a context off of a thread or put a context on a thread. Now multiple requests for the same transaction are no longer required to run on the same thread, but this requires management of mapping users to Contexts.

Another problem results once you have a Context set up on a thread and now you want to move to a different thread. With databases that require that multiple requests for the same transaction run on the same thread, it is not possible to move to a different thread, while with other database programs it is possible to move to a different thread.

Servers often have to support programs and services which have multiple ways of handling Context. A need exists for a flexible mechanism to allow all of these programs/services to coexist and yet have the execution thread have the proper Context set up.

*Col. 2, ll 1-25.*

... The general purpose of a ContextControl object 140 of FIG. 1C is to manage context relative to the lifecycle of an individual thread 138, and manage context across multiple threads 138 when those threads are executing requests which are associated with one another. It is understood that services will deal with context relative to threads 138 in one of two ways. A first type of services requires that the request is moved to the thread 138 which has a specific context on it. This first type of services are not capable of moving their context from one thread to another. A second type of services 134 can add context to the thread 138 where the request is executing. Since the context can be added to any thread 138, it can also be moved from one thread to another.

In accordance with features of the preferred embodiment, ContextControl Objects 140 are used to create and destroy context on threads 138 for both the first type of service, where multiple requests for the same transaction run on the same thread, and the second type of service, where multiple requests for the same transaction can run on different threads. In addition, to move from one thread 138 to another thread 138, the ContextControl Objects 140 are used to remove context from the one thread 138 and to add context to the other thread 138 for the second type of service.

To put onto a thread 138 means making the context information accessible to services executing on a given thread 138, in the format familiar to the service, such as an object service 134. To take off of a thread means removing the context information from a given thread 138. Resume is the process of associating/reassociating a context with the current execution thread. Suspend is the process of disassociating a context from the current execution thread.

As illustrated in FIG. 1C, the ContextControl Objects 140 includes six methods shown in all capital letters in FIG. 1C, while a mixed case naming convention is used where for method names composed of more than one word, the first letter of words after the first word are capitalized, as follows:

1. init() 142;

2. uninit 144;
3. getAndSuspendContext(ABCContextHandleList) 146;
4. setAndResumeContext(ABCContextHandleList, ServiceContext) 148;
5. copyContext(ABCContextHandleList) 150; and
6. end Association(ABCContextHandleList) 152.

*Col. 3, line 55, through col. 4, line 30.*

As understood by Applicants, Hutchinson fails to teach or suggest "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete."

Again, in claim 13, the (1) component's state is discarded ("discarding the processing state of the component"), the (2) component indicated that work is complete ("responsive to the component indicating completion of the work") and the (3) client of the component did not indicate that the component's work is complete ("before receiving any indication from the client that the component's work is complete"). Applicants respectfully submit that these three elements have not been shown in the Office Action.

For at least this reason, claim 13 should be allowed. Such action is respectfully requested.

**Claim 18**

Claim 18 is generally directed to a system service comprising "destroying the processing state of the application component responsive to the indication from the application component that processing by the application component of the work is complete and without action from the client program."

In claim 18, the (1) component's processing state is destroyed ("destroying the processing state of the application component), the (2) component indicated that work is complete ("responsive to the indication from the application component that processing by the application component of the work is complete") and the (3) client of the component did not take action ("without action from the client program"). Applicants respectfully submit that these three elements have not been shown in the Office Action. This language is not recited in Helland, nor is it taught or suggested by Hutchinson. For at least this reason, claim 18 should be allowed. Such action is respectfully requested.

**Claim 19**

Claim 19 depends from claim 18. Since claim 19 depends from claim 18, it should be allowed for at least the reasons stated for claim 18. In view of the foregoing discussion claim 18, the merits of the separate patentability of dependent claim 19 not belabored at this time. Claims 19 should be allowed. Such action is respectfully requested.

**Claim 20**

Claim 20 depends from claim 18. Since claim 20 depend from claim 18, it should be allowed for at least the reasons stated for claim 18. In view of the foregoing discussion claim 18, the merits of the separate patentability of dependent claim 20 is not belabored at this time. Claim 20 should be allowed. Such action is respectfully requested.

**Claim 21**

Claim 21 is generally directed to a method of enhancing scalability of server applications comprising an "application component having state ... destroying the state by the operating service in response to an indication from the application component without action by the client."

In claim 21, the (1) application component's state is destroyed ("destroying the state by the operating service"), the (2) destruction is responsive to the application component ("in response to an indication from the application component") and the (3) client did not take action ("without action by the client"). Applicants respectfully submit that these three elements have not been shown in the Office Action. This language is not recited in Helland, nor is it taught or suggested by Hutchinson. For at least this reason, claim 18 should be allowed. Such action is respectfully requested.

## CONCLUSION

The claims in their present form should now be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

By _____
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446